º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Alberto Jesús Durán López
Grupo de prácticas: 1
Fecha de entrega:
Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz

Sistema operativo utilizado: *Ubuntu 14.04.1 - 64 bits*

Versión de gcc utilizada: gcc version 4.9.4 (gcc -v)

Adjunte el contenido del fichero /proc/cpuinfo de la máquina en la que ha tomado las medidas

1. Para el núcleo que se muestra en la Figura 1 (ver guion de prácticas), y para un programa que implemente la multiplicación de matrices (use variables globales):

   1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos (use –O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

   1.2 Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.

   1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

A) MULTIPLICACIÓN DE MATRICES:
CÓDIGO FUENTE: `pmm-secuencial.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define MAX 3000

int **matrizA, **matrizB, **matrizC;

int main(int argc, char** argv)
{

  int i, j, k;
  int valor;
  int suma=0;
```

```
  if (argc<2){
    printf("Introduzca tamaño de la matriz\n");
    exit(-1);
  }

  int N = atoi(argv[1]);

  if(N>MAX)
     N=MAX;

  matrizA=(int **)malloc(N*sizeof(int*));
  matrizB=(int **)malloc(N*sizeof(int*));
  matrizC=(int **)malloc(N*sizeof(int*));

  for(i=0; i<N; i++){
    matrizA[i]=(int*)malloc(N*sizeof(int));
    matrizB[i]=(int*)malloc(N*sizeof(int));
    matrizC[i]=(int*)malloc(N*sizeof(int));
  }

  srand (time(NULL));

  for (i=0; i<N; i++){
    for(j=0; j<N; j++){
      valor=rand() % 1000; //Número aleatorio entre 0 y 999
      matrizA[i][j]=valor;
      matrizB[j][i]=valor;
    }
  }


  struct timespec cgt1,cgt2; double ncgt;

  clock_gettime(CLOCK_REALTIME,&cgt1);

  //Calculamos el producto de las matrices
  for (i=0; i<N; i++){
    for(j=0; j<N; j++){
        for (k=0; k<N; k++){
          matrizC[i][j]+= matrizA[i][k]*matrizB[k][j];
                    }
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);

  //Calculamos la diferencia
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));


  printf("Tiempo(seg.):%f\t / Tamaño:%u\t/ \n", ncgt, N);

  //Liberamos memoria
  for(i=0; i<N; i++){
    free(matrizA[i]);
    free(matrizB[i]);
    free(matrizC[i]);
  }

  free(matrizA);
```

```
  free(matrizB);
  free(matrizC);

  return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Realizo un cambio en la ejecución de los bucles for anidados. En vez de hacer (i,j,k) hago (i,k,j) para acceder a elementos consecutivos al llamar a MatrizB[k][j]

Modificación b) –explicación-: He realizado un desenrollado del bucle, realizando saltos de 10 iteraciones para reducirlas.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmm-secuencial-modificado_a.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define MAX 3000

int **matrizA, **matrizB, **matrizC;

int main(int argc, char** argv)
{

  int i, j, k;
  int valor;
  int suma=0;

  if (argc<2){
    printf("Introduzca tamaño de la matriz\n");
    exit(-1);
  }

  int N = atoi(argv[1]);

  if(N>MAX)
     N=MAX;

  matrizA=(int **)malloc(N*sizeof(int*));
  matrizB=(int **)malloc(N*sizeof(int*));
  matrizC=(int **)malloc(N*sizeof(int*));

  for(i=0; i<N; i++){
    matrizA[i]=(int*)malloc(N*sizeof(int));
    matrizB[i]=(int*)malloc(N*sizeof(int));
    matrizC[i]=(int*)malloc(N*sizeof(int));
  }

  srand (time(NULL));

  for (i=0; i<N; i++){
    for(j=0; j<N; j++){
      valor=rand() % 1000; //Número aleatorio entre 0 y 999
      matrizA[i][j]=valor;
      matrizB[j][i]=valor;
    }

  }
```

```
  struct timespec cgt1,cgt2; double ncgt;

  clock_gettime(CLOCK_REALTIME,&cgt1);

  //Calculamos el producto de las matrices
  for (i=0; i<N; i++){
    for(k=0; k<N; k++){
        for (j=0; j<N; j++){
          matrizC[i][j]+= matrizA[i][k]*matrizB[k][j];
                   }
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);

  //Calculamos la diferencia
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));


  printf("Tiempo(seg.):%f\t / Tamaño:%u\t/ \n", ncgt, N);

  //Liberamos memoria
  for(i=0; i<N; i++){
    free(matrizA[i]);
    free(matrizB[i]);
    free(matrizC[i]);
  }

  free(matrizA);
  free(matrizB);
  free(matrizC);

  return 0;
}
```

b) `pmm-secuencial-modificado_b.c`

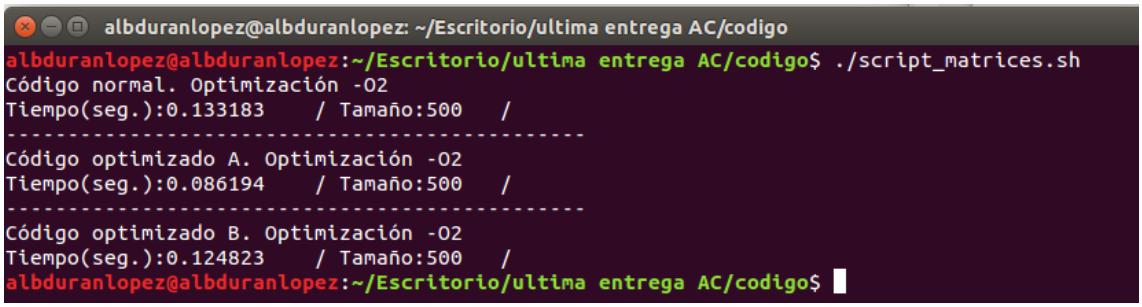
```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define MAX 3000

int **matrizA, **matrizB, **matrizC;

int main(int argc, char** argv)
{

  int i, j, k;
  int valor;
  int suma=0;

  if (argc<2){
    printf("Introduzca tamaño de la matriz\n");
    exit(-1);
  }

  int N = atoi(argv[1]);
```

```
  if(N%10!=0) {
   fprintf(stderr,"Num debe ser divisible entre 10\n");
   exit(-1);
 }

  if(N>MAX)
     N=MAX;

  matrizA=(int **)malloc(N*sizeof(int*));
  matrizB=(int **)malloc(N*sizeof(int*));
  matrizC=(int **)malloc(N*sizeof(int*));

  for(i=0; i<N; i++){
    matrizA[i]=(int*)malloc(N*sizeof(int));
    matrizB[i]=(int*)malloc(N*sizeof(int));
    matrizC[i]=(int*)malloc(N*sizeof(int));
  }

  srand (time(NULL));

  for (i=0; i<N; i++){
    for(j=0; j<N; j++){
      valor=rand() % 1000; //Número aleatorio entre 0 y 999
      matrizA[i][j]=valor;
      matrizB[j][i]=valor;
    }

  }

  int t0,t1,t2,t3,t4,t5,t6,t7,t8,t9;

  struct timespec cgt1,cgt2; double ncgt;

  clock_gettime(CLOCK_REALTIME,&cgt1);

  //Calculamos el producto de las matrices
  for (i=0; i<N; i++){
    for(j=0; j<N; j++){
        t0=t1=t2=t3=t4=t5=t6=t7=t8=t9=0;
        for (k=0; k<N; k+=10){
          t0+=matrizA[i][k]*matrizB[k][j];
          t1+=matrizA[i][k+1]*matrizB[k+1][j];
          t2+=matrizA[i][k+2]*matrizB[k+2][j];
          t3+=matrizA[i][k+3]*matrizB[k+3][j];
          t4+=matrizA[i][k+4]*matrizB[k+4][j];
          t5+=matrizA[i][k+5]*matrizB[k+5][j];
          t6+=matrizA[i][k+6]*matrizB[k+6][j];
          t7+=matrizA[i][k+7]*matrizB[k+7][j];
          t8+=matrizA[i][k+8]*matrizB[k+8][j];
          t9+=matrizA[i][k+9]*matrizB[k+9][j];

       }
         matrizC[i][j]=t0+t1+t2+t3+t4+t5+t6+t7+t8+t9;
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);

  //Calculamos la diferencia
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
```

```
  printf("Tiempo(seg.):%f\t / Tamaño:%u\t/ \n", ncgt, N);

  //Liberamos memoria
  for(i=0; i<N; i++){
    free(matrizA[i]);
    free(matrizB[i]);
    free(matrizC[i]);
  }

  free(matrizA);
  free(matrizB);
  free(matrizC);

  return 0;
}
```

Capturas de pantalla (que muestren que el resultado es correcto):

Aunque no lo pedían, he realizado un script para mostrar los resultados para un N=500 y optimización -02. Así que en una única captura mostraré todos los resultados de los tiempos.



1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.133183*
Modificación a)	0.086194
Modificación b)	0.124823

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Podemos observar claramente que el código sin modificar obtiene peores tiempos que las modificaciones realizadas con la optimización pedida -02. En este caso, la modificación a) es la que tiene el mejor tiempo de todos, 0.086194 segundos en realizar el producto de dos matrices 500*500. Es decir, realizar un cambio en la ejecución de los bucles for anidados. En vez de hacer (i,j,k), hacer(i,k,j).

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

pmm-secuencial.s

```
call            clock_gettime
                movq
                matrizC(%rip), %r13
                movq
                matrizA(%rip), %r12
                movq
                matrizB(%rip), %r11
.L11:
                movq
                0(%r13,%rbp,8),
%rdx
                movq
                (%r12,%rbp,8), %r10
                xorl
                %r8d, %r8d
                .p2align 4,,10
                .p2align 3
.L14:
                movl
                (%rdx,%r8,4), %esi
                leaq
                0(,%r8,4), %r9
                xorl
                %eax, %eax
                .p2align 4,,10
                .p2align 3
.L9:
                movq
                (%r11,%rax,8), %rdi
                movl
                (%r10,%rax,4), %ecx
                addq
                $1, %rax
                imull
                (%rdi,%r9), %ecx
                addl
                %ecx, %esi
                cmpl
                %eax, %ebx
                movl
                %esi, (%rdx,%r8,4)
                jg
                .L9
                addq
                $1, %r8
                cmpl
                %r8d, %ebx
                jg
                .L14
                addq
                $1, %rbp
                cmpl
                %ebp, %ebx
                jg
                .L11
                leaq
                16(%rsp), %rsi
                xorl
                %edi, %edi
                xorl
                %ebp, %ebp
                call
                clock_gettime
```

pmm-secuencial-modificado_a.s

```
call            clock_gettime
                movq
                matrizC(%rip),
%rbp
                movq
                matrizA(%rip),
%r11
                xorl
                %r10d, %r10d
                movq
                matrizB(%rip),
%r9
.L11:
                movq
                0(%rbp,%r10,8),
%rsi
                movq
                (%r11,%r10,8),
%r8
                xorl
                %edi, %edi
                .p2align 4,,10
                .p2align 3
.L14:
                movq
                (%r9,%rdi,8),
%rdx
                xorl
                %eax, %eax
                .p2align 4,,10
                .p2align 3
.L9:
                movl
                (%r8,%rdi,4),
%ecx
                imull
                (%rdx,%rax,4),
%ecx
                addl
                %ecx, (%rsi,
%rax,4)
                addq
                $1, %rax
                cmpl
                %eax, %ebx
                jg
                .L9
                addq
                $1, %rdi
                cmpl
                %edi, %ebx
                jg
                .L14
                addq
                $1, %r10
                cmpl
                %r10d, %ebx
                jg
                .L11
                leaq
                16(%rsp), %rsi
                xorl
                %edi, %edi
```

pmm-secuencial-modificado_b.s

```
call            clock_gettime
                movq
                matrizA(%rip), %rax
                movq
                $0, 32(%rsp)
                movq
                %rax, 40(%rsp)
                movq
                matrizB(%rip), %rax
                movq
                %rax, 24(%rsp)
.L12:
                movq
                40(%rsp), %rax
                movq
                32(%rsp), %rbx
                movq
                $0, 8(%rsp)
                movq
                (%rax,%rbx,8), %rax
                movq
                %rax, 16(%rsp)
                .p2align 4,,10
                .p2align 3
.L15:
                movq
                8(%rsp), %rax
                movq
                16(%rsp), %rdx
                xorl
                %edi, %edi
                xorl
                %r15d, %r15d
                xorl
                %r14d, %r14d
                xorl
                %r13d, %r13d
                xorl
                %r12d, %r12d
                xorl
                %ebp, %ebp
                xorl
                %ebx, %ebx
                leaq
                0(,%rax,4), %rcx
                movq
                24(%rsp), %rax
                xorl
                %r11d, %r11d
                xorl
                %r10d, %r10d
                xorl
                %r9d, %r9d
                xorl
                %r8d, %r8d
                movl
                %edi, (%rsp)
                .p2align 4,,10
                .p2align 3
.L10:
                movq
                (%rax), %rsi
                movl
```

```
                movq
                16(%rsp), %rax
                subq
                (%rsp), %rax
                movl
                %ebx, %edx
                pxor
                %xmm1, %xmm1
                movl
                $.LC2, %esi
                pxor
                %xmm0, %xmm0
                movl
                $1, %edi
                cvtsi2sdq
                %rax, %xmm1
                movq
                24(%rsp), %rax
                subq
                8(%rsp), %rax
                cvtsi2sdq
                %rax, %xmm0
                movl
                $1, %eax
                divsd
                .LC1(%rip), %xmm0
                addsd
                %xmm1, %xmm0
                call
                __printf_chk
.L13:
                movq
                matrizA(%rip), %rax
                movq
                (%rax,%rbp,8), %rdi
                call
                free
                movq
                matrizB(%rip), %rax
                movq
                (%rax,%rbp,8), %rdi
                call
                free
                movq
                matrizC(%rip), %rax
                movq
                (%rax,%rbp,8), %rdi
                addq
                $1, %rbp
                call
                free
                cmpl
                %ebp, %ebx
                jg
                .L13
.L15:
                movq
                matrizA(%rip), %rdi
                call
                free
                movq
                matrizB(%rip), %rdi
                call
                free
                movq
                matrizC(%rip), %rdi
                call
                free
                addq
                $40, %rsp
                .cfi_remember_state
                .cfi_def_cfa_offset
56
                xorl
                %eax, %eax
                popq
                %rbx
                .cfi_def_cfa_offset
48
                xorl
                %ebp, %ebp
                call
                clock_gettime
                (%rdx), %edi
                addl
                $10, %r8d
                addq
                $40, %rdx
                addq
                $80, %rax
                imull
                (%rsi,%rcx), %edi
                movq            -
72(%rax), %rsi
                addl
                %edi, (%rsp)
                movl            -
36(%rdx), %edi
                imull
                (%rsi,%rcx), %edi
                movq            -
64(%rax), %rsi
                addl
                %edi, %r9d
                movl            -
32(%rdx), %edi
                imull
                (%rsi,%rcx), %edi
                movq            -
56(%rax), %rsi
                addl
                %edi, %r10d
                movl            -
28(%rdx), %edi
                imull
                (%rsi,%rcx), %edi
                movq            -
48(%rax), %rsi
                addl
                %edi, %r11d
                movl            -
24(%rdx), %edi
                imull
                (%rsi,%rcx), %edi
                movq            -
40(%rax), %rsi
                addl
                %edi, %ebx
                movl            -
20(%rdx), %edi
                imull
                (%rsi,%rcx), %edi
                movq            -
32(%rax), %rsi
                addl
                %edi, %ebp
                movl            -
16(%rdx), %edi
                imull
                (%rsi,%rcx), %edi
                movq            -
24(%rax), %rsi
                addl
                %edi, %r12d
                movl            -
12(%rdx), %edi
                imull
                (%rsi,%rcx), %edi
                movq            -
16(%rax), %rsi
                addl
                %edi, %r13d
                movl            -
8(%rdx), %edi
                imull
                (%rsi,%rcx), %edi
                movq            -
8(%rax), %rsi
                addl
                %edi, %r14d
                movl            -
4(%rdx), %edi
```

```
                popq
                %rbp
                .cfi_def_cfa_offset
40
                popq
                %r12
                .cfi_def_cfa_offset
32
                popq
                %r13
                .cfi_def_cfa_offset
24
                popq
                %r14
                .cfi_def_cfa_offset
16
                popq
                %r15
                .cfi_def_cfa_offset
8
                ret
.L3:
                .cfi_restore_state
                xorl
                %edi, %edi
                call
                time
                movl
                %eax, %edi
                call
                srand
                movq
                %rsp, %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

```
                imull
                (%rsi,%rcx), %edi
                addl
                %edi, %r15d
                cmpl
                4(%rsp), %r8d
                jl
                .L10
                movl
                (%rsp), %edi
                movq
                8(%rsp), %rax
                addl
                %r9d, %edi
                addl
                %edi, %r10d
                addl
                %r10d, %r11d
                addl
                %r11d, %ebx
                addl
                %ebx, %ebp
                movq
                16(%rsp), %rbx
                addl
                %ebp, %r12d
                addl
                %r12d, %r13d
                addl
                %r13d, %r14d
                addl
                %r14d, %r15d
                movl
                %r15d, (%rbx,
%rax,4)
                addq
                $1, %rax
                cmpl
                %eax, 4(%rsp)
                movq
                %rax, 8(%rsp)
                jg
                .L15
                addq
                $1, 32(%rsp)
                movq
                32(%rsp), %rax
                cmpl
                %eax, 4(%rsp)
                jg
                .L12
                leaq
                64(%rsp), %rsi
                xorl
                %edi, %edi
                xorl
                %ebx, %ebx
                call
                clock_gettime
                movq
                64(%rsp), %rax
                subq
                48(%rsp), %rax
                movl
                $.LC3, %esi
                pxor
                %xmm1, %xmm1
                movl
                4(%rsp), %ebp
                pxor
                %xmm0, %xmm0
                movl
                $1, %edi
                cvtsi2sdq
                %rax, %xmm1
                movq
                72(%rsp), %rax
                subq
                56(%rsp), %rax
```

```
                movl
                %ebp, %edx
                cvtsi2sdq
                %rax, %xmm0
                movl
                $1, %eax
                divsd
                .LC2(%rip), %xmm0
                addsd
                %xmm1, %xmm0
                call
                __printf_chk
.L14:
                movq
                matrizA(%rip), %rax
                movq
                (%rax,%rbx,8), %rdi
                call
                free
                movq
                matrizB(%rip), %rax
                movq
                (%rax,%rbx,8), %rdi
                call
                free
                movq
                matrizC(%rip), %rax
                movq
                (%rax,%rbx,8), %rdi
                addq
                $1, %rbx
                call
                free
                cmpl
                %ebx, %ebp
                jg
                .L14
.L16:
                movq
                matrizA(%rip), %rdi
                call
                free
                movq
                matrizB(%rip), %rdi
                call
                free
                movq
                matrizC(%rip), %rdi
                call
                free
                addq
                $88, %rsp
                .cfi_remember_state
                .cfi_def_cfa_offset
56
                xorl
                %eax, %eax
                popq
                %rbx
                .cfi_def_cfa_offset
48
                popq
                %rbp
                .cfi_def_cfa_offset
40
                popq
                %r12
                .cfi_def_cfa_offset
32
                popq
                %r13
                .cfi_def_cfa_offset
24
                popq
                %r14
                .cfi_def_cfa_offset
16
                popq
                %r15
```

		.cfi_def_cfa_offset 8 ret .L4: .cfi_restore_state xorl %edi, %edi call time movl %eax, %edi call srand leaq 48(%rsp), %rsi xorl %edi, %edi call clock_gettime

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-original.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

struct {
  int a;
  int b;
} s[5000];


int  main(){

  int MAX=40000;
  int ii, i;
  int X1, X2;
  int R[MAX];

  srand(time(NULL));

  for(i=0; i<5000; i++){
     s[i].a=rand()%5000;
     s[i].b=rand()%5000;
  }

  struct timespec cgt1,cgt2;
  double ncgt;

  clock_gettime(CLOCK_REALTIME,&cgt1);

  for (ii=0; ii<40000;ii++){
    X1=0; X2=0;
    for(i=0; i<5000;i++)
      X1+=2*s[i].a+ii;
    for(i=0; i<5000;i++)
      X2+=3*s[i].b-ii;
    if (X1<X2)
      R[ii]=X1;
    else
      R[ii]=X2;
  }
```

```
  clock_gettime(CLOCK_REALTIME,&cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

  printf("\nTiempo (seg.) = %11.9f\n", ncgt);
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Junto los dos bucles for en sólo uno ya que al fin y al cabo el bucle hace las mismas iteraciones y hace lo mismo.

Modificación b) –explicación-: Además de la modificación a) anterior, realizo un desenrollado del bucle para así decrementar el número de iteraciones realizadas y disminuir su tiempo de ejecución.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figura1**`-modificado_a.c`**

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

struct {
  int a;
  int b;
} s[5000];

int  main(){

  int MAX=40000;
  int ii, i;
  int X1, X2;
  int R[MAX];

  srand(time(NULL));

  for(i=0; i<5000; i++){
     s[i].a=rand()%5000;
     s[i].b=rand()%5000;
  }

  struct timespec cgt1,cgt2;
  double ncgt;

  clock_gettime(CLOCK_REALTIME,&cgt1);

//Agrupo los dos for en uno solo
  for (ii=0; ii<40000;ii++){
    X1=0; X2=0;
    for(i=0; i<5000;i++){
      X1+=2*s[i].a+ii;
      X2+=3*s[i].b-ii;
    }
    if (X1<X2)
      R[ii]=X1;
    else
      R[ii]=X2;
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
```

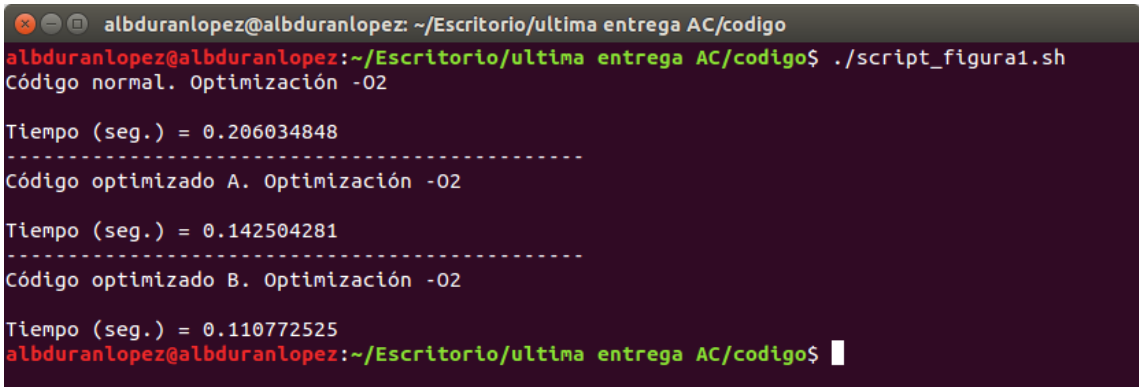
```
cgt1.tv_nsec)/(1.e+9));

  printf("\nTiempo (seg.) = %11.9f\n", ncgt);
}
```

b) figura1**-modificado_b.c**

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

struct {
  int a;
  int b;
} s[5000];

int  main(){

  int MAX=40000;
  int ii, i;
  int X1, X2;
  int R[MAX];

  srand(time(NULL));

  for(i=0; i<5000; i++){
     s[i].a=rand()%5000;
     s[i].b=rand()%5000;
  }

  struct timespec cgt1,cgt2;
  double ncgt;

  clock_gettime(CLOCK_REALTIME,&cgt1);

//Agrupo los dos for en uno solo y descompongo el bucle for
  for (ii=0; ii<40000;ii++){
    X1=0; X2=0;
    for(i=0; i<5000;i+=10){
      X1+=2*s[i].a+ii;
      X2+=3*s[i].b-ii;
      X1+=2*s[i+1].a+ii;
      X2+=3*s[i+1].b-ii;
      X1+=2*s[i+2].a+ii;
      X2+=3*s[i+2].b-ii;
      X1+=2*s[i+3].a+ii;
      X2+=3*s[i+3].b-ii;
      X1+=2*s[i+4].a+ii;
      X2+=3*s[i+4].b-ii;
      X1+=2*s[i+5].a+ii;
      X2+=3*s[i+5].b-ii;
      X1+=2*s[i+6].a+ii;
      X2+=3*s[i+6].b-ii;
      X1+=2*s[i+7].a+ii;
      X2+=3*s[i+7].b-ii;
      X1+=2*s[i+8].a+ii;
      X2+=3*s[i+8].b-ii;
      X1+=2*s[i+9].a+ii;
      X2+=3*s[i+9].b-ii;
    }
```

```
    if (X1<X2)
      R[ii]=X1;
    else
      R[ii]=X2;
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

  printf("\nTiempo (seg.) = %11.9f\n", ncgt);
}
```

Capturas de pantalla (que muestren que el resultado es correcto):



1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	0.206034848
Modificación a)	0.142504281
Modificación b)	0.110772525

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Como podemos observar, el código sin modificar se ejecuta en un tiempo mayor que los otros dos modificados. Es por esto por lo que podemos observar que las modificaciones realizadas de unir los dos bucles for en uno son efectivas. Comparando la segunda modificación con la tercera, vemos que la modificación b) de desenrollado del bucle for tiene un tiempo menor por lo que también resulta efectiva dicha modificación

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

Figura1-original.s	Figura1-modificado_a.s	Figura1-modificado_b.s
call clock_gettime xorl %r8d, %r8d .p2align 4,,10 .p2align 3	call clock_gettime xorl %r8d, %r8d .p2align 4,,10 .p2align 3	call clock_gettime xorl %r11d, %r11d .p2align 4,,10 .p2align 3

```
.L3:
                movl
                %r8d, %edi
                movl
                $s, %eax
                xorl
                %esi, %esi
                .p2align 4,,10
                .p2align 3
.L4:
                movl
                (%rax), %edx
                addq
                $8, %rax
                leal
                (%rdi,%rdx,2),
%edx
                addl
                %edx, %esi
                cmpq
                %rax, %r12
                jne
                .L4
                movl
                $s+4, %eax
                xorl
                %ecx, %ecx
                .p2align 4,,10
                .p2align 3
.L5:
                movl
                (%rax), %edx
                addq
                $8, %rax
                leal
                (%rdx,%rdx,2),
%edx
                subl
                %edi, %edx
                addl
                %edx, %ecx
                cmpq
                $s+40004, %rax
                jne
                .L5
                cmpl
                %esi, %ecx
                cmovg
                %esi, %ecx
                movl
                %ecx, (%rbx,
%r8,4)
                addq
                $1, %r8
                cmpq
                $40000, %r8
                jne
                .L3
                leaq
                -64(%rbp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

```
.L2:
                movl
                %r8d, %edi
                movl
                $s, %eax
                xorl
                %ecx, %ecx
                xorl
                %esi, %esi
                .p2align 4,,10
                .p2align 3
.L3:
                movl
                (%rax), %edx
                addq
                $8, %rax
                leal
                (%rdi,%rdx,2),
%edx
                addl
                %edx, %esi
                movl
                -4(%rax), %edx
                leal
                (%rdx,%rdx,2),
%edx
                subl
                %edi, %edx
                addl
                %edx, %ecx
                cmpq
                $s+40000, %rax
                jne
                .L3
                cmpl
                %esi, %ecx
                cmovg
                %esi, %ecx
                movl
                %ecx, (%rbx,
%r8,4)
                addq
                $1, %r8
                cmpq
                $40000, %r8
                jne
                .L2
                leaq
                -48(%rbp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

```
.L3:
                movl
                %r11d, %r10d
                movl            $s,
%eax
                xorl
                %edx, %edx
                xorl
                %esi, %esi
                .p2align 4,,10
                .p2align 3
.L4:
                movl
                (%rax), %ecx
                addq            $80,
%rax
                leal
                (%r10,%rcx,2), %r13d
                movl            -
76(%rax), %ecx
                addl
                %esi, %r13d
                leal
                (%rcx,%rcx,2), %esi
                movl            -
72(%rax), %ecx
                subl
                %r10d, %esi
                addl
                %esi, %edx
                leal
                (%r10,%rcx,2), %esi
                movl            -
68(%rax), %ecx
                addl
                %esi, %r13d
                leal
                (%rcx,%rcx,2), %ecx
                subl
                %r10d, %ecx
                leal
                (%rcx,%rdx), %esi
                movl            -
64(%rax), %edx
                leal
                (%r10,%rdx,2), %edx
                addl
                %edx, %r13d
                movl            -
60(%rax), %edx
                leal
                (%rdx,%rdx,2), %edx
                subl
                %r10d, %edx
                leal
                (%rdx,%rsi), %ecx
                movl            -
56(%rax), %edx
                leal
                (%r10,%rdx,2), %r9d
                movl            -
52(%rax), %edx
                addl
                %r9d, %r13d
                leal
                (%rdx,%rdx,2), %r9d
                subl
                %r10d, %r9d
                leal
                (%r9,%rcx), %edx
                movl            -
48(%rax), %ecx
                leal
                (%r10,%rcx,2), %r9d
                movl            -
44(%rax), %ecx
                addl
                %r9d, %r13d
                leal
```

		(%rcx,%rcx,2), %r8d subl %r10d, %r8d leal (%r8,%rdx), %r9d movl - 40(%rax), %edx leal (%r10,%rdx,2), %r8d movl - 36(%rax), %edx addl %r8d, %r13d leal (%rdx,%rdx,2), %edi movl - 32(%rax), %edx subl %r10d, %edi leal (%rdi,%r9), %r8d leal (%r10,%rdx,2), %r9d movl - 28(%rax), %edx addl %r13d, %r9d leal (%rdx,%rdx,2), %esi movl - 24(%rax), %edx subl %r10d, %esi leal (%rsi,%r8), %edi leal (%r10,%rdx,2), %r8d movl - 20(%rax), %edx addl %r9d, %r8d leal (%rdx,%rdx,2), %ecx movl - 16(%rax), %edx subl %r10d, %ecx leal (%rcx,%rdi), %esi leal (%r10,%rdx,2), %edi movl - 12(%rax), %edx addl %r8d, %edi leal (%rdx,%rdx,2), %edx subl %r10d, %edx leal (%rdx,%rsi), %ecx movl - 8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12 jne .L4 cmpl

		%esi, %edx cmovg %esi, %edx movl %edx, (%rbx,%r11,4) addq $1, %r11 cmpq $40000, %r11 jne .L3 leaq -64(%rbp), %rsi xorl %edi, %edi call clock_gettime

2. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina denominada DAXPY (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

```
for (i=1;i<=N,i++) y[i]= a*x[i] + y[i];
```

2.1. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O0, -O2, -O3) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo del número de operaciones en coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

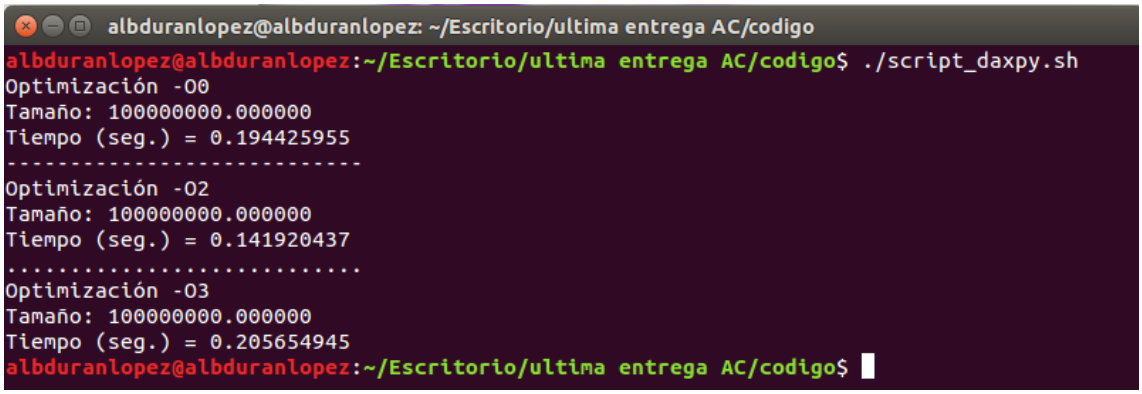
CÓDIGO FUENTE: daxpy.c
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>


int main(int argc, char *argv[]){

    double N=100000000; //Tamaño del vector
    double a = 10, i; //Constante
    double *x, *y;

    x = (double*) malloc(N*sizeof(int));
    y = (double*) malloc(N*sizeof(int));


    for (i=0; i<N; i++){
        x[i] = i*5;
        y[i] = i/2+8;
```

```
    }

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);
    //Becnhmark Linpack
    for (i=0; i<N; i++)
        y[i] += a*x[i];

    clock_gettime(CLOCK_REALTIME,&cgt2);

    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

    printf("Tamaño: %i", N);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    free(x);
    free(y);

    return 0;
}
```

Tiempos ejec.	**-O0**	**-O2**	**-O3**
	0.194425955	0.141920437	0.205654945

CAPTURAS DE PANTALLA:



COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

-O0: Desconecta por completo la optimización del código y es el predeterminado si no se especifica ningún -O. Es por esto por lo que el código en ensamblador de daxpyO0.c es más grande que el resto.

-O2: suele ser la mejor opción de optimización, además, es el más recomendado. Se aumenta el rendimiento del código sin comprometer el tamaño y sin tomar mucho más tiempo de compilación.

-O3: Aplica una optimización más agresiva pero puede defraudar. Activa optimizaciones que son caras en términos de tiempo de compilación y uso de memoria. No garantiza una mejora del rendimiento ya que puede ralentizar el sistema por lo que no se recomienda usarlo.

CÓDIGO EN ENSAMBLADOR (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpyO0.s

```
call            clock_gettime
                movl
                $0, %ebx
                jmp
                .L8
.L9:
                movslq
                %ebx, %rax
                leaq
                0(,%rax,8), %rdx
                movq            -
64(%rbp), %rax
                addq
                %rax, %rdx
                movslq
                %ebx, %rax
                leaq
                0(,%rax,8), %rcx
                movq            -
64(%rbp), %rax
                addq
                %rcx, %rax
                movsd
                (%rax), %xmm1
                movslq
                %ebx, %rax
                leaq
                0(,%rax,8), %rcx
                movq            -
72(%rbp), %rax
                addq
                %rcx, %rax
                movsd
                (%rax), %xmm0
                mulsd           -
80(%rbp), %xmm0
                addsd
                %xmm0, %xmm1
                movq
                %xmm1, %rax
                movq
                %rax, (%rdx)
                addl
                $1, %ebx
.L8:
                pxor
                %xmm0, %xmm0
                cvtsi2sd
                %ebx, %xmm0
                movsd           -
88(%rbp), %xmm1
                ucomisd
                %xmm0, %xmm1
                ja
                .L9
                leaq            -
32(%rbp), %rax
                movq
                %rax, %rsi
                movl
                $0, %edi
```

daxpyO2.s

```
call            clock_gettime
                movsd
                .LC1(%rip),
%xmm1
                xorl
                %eax, %eax
                .p2align 4,,10
                .p2align 3
.L4:
                movsd
                (%rbx,%rax,8),
%xmm0
                leal
                1(%rax), %edx
                movsd
                .LC0(%rip),
%xmm3
                mulsd
                %xmm1, %xmm0
                addsd
                (%r12,%rax,8),
%xmm0
                movsd
                %xmm0,
(%r12,%rax,8)
                pxor
                %xmm0, %xmm0
                addq
                $1, %rax
                cvtsi2sd
                %edx, %xmm0
                ucomisd
                %xmm0, %xmm3
                ja
                .L4
                leaq
                32(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

daxpyO3.s

```
call            clock_gettime
                movsd
                .LC1(%rip),
%xmm1
                xorl
                %eax, %eax
                movsd
                .LC0(%rip),
%xmm3
                .p2align 4,,10
                .p2align 3
.L4:
                movsd
                (%rbx,%rax,8),
%xmm0
                leal
                1(%rax), %edx
                mulsd
                %xmm1, %xmm0
                addsd
                (%r12,%rax,8),
%xmm0
                movsd
                %xmm0,
(%r12,%rax,8)
                pxor
                %xmm0, %xmm0
                addq
                $1, %rax
                cvtsi2sd
                %edx, %xmm0
                ucomisd
                %xmm0, %xmm3
                ja
                .L4
                leaq
                32(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

`call clock_gettime`		